AD-A234 898

COORDINATION IN LARGE SCALE SOFTWARE DEVELOPMENT

Robert E. Kraut and Lynn A. Streeter
Bellcore
Morristown, NJ 07962
USA

NRL

ABSTRACT

Successful software development requires tight coordination among subgroups involved in the development process. Coordination is difficult because of the division of labor, interdependence and uncertainty inherent in large software projects. A survey in 65 software development projects reveals that informal communication is necessary for coordination under these circumstances. Results show that software professionals got much of their information directly from other people. They perceived that interpersonal techniques for getting information from beyond their immediate workgroup were underused, while more formal procedures for tracking routine information were overused compared to their value. Technically uncertain projects and highly interdependent ones had staffs who were poorly informed and were poorly coordinated. When project members had a large network of personal contacts outside the project, information flow improved, especially when the project was uncertain. The paper concludes with organizational and technological suggestions for increasing the flow of relevant information across functional boundaries in projects.

DTIC
ELECTE
APR 2 4 1991
S D
D

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

91 4 05 063

COORDINATION IN SOFTWARE DEVELOPMENT

Robert E. Kraut and Lynn A. Streeter
Bellcore
Morristown, NJ 07962
USA

INTRODUCTION

Since its inception, the software industry has been in crisis. As Blazer (1975) noted fifteen years ago, "[Software] is unreliable, delivered late, unresponsive to change, inefficient, and expensive," "... and has been for the past 20 years." In a survey of software contractors and government contract officers, over half of the respondents believed that calendar overruns, cost overruns, code that required in-house modifications before being usable, and code that was difficult to modify were common problems in the software projects they supervised (U.S. Department of Commerce, 1979). Even today problems with large software systems are common and highly publicized occurrences (e.g.,*Washington Post*, 1990; Burnham, 1990; Travis, 1990).

The problem of coordinating activities while developing large software systems is a major contributor to this continued crisis. Coordination is the direction of "individuals' efforts toward achieving common and explicitly recognized goals" (Blau and Scott, 1962) and "the integration or linking together of different parts of an organization to accomplish a collective set of tasks" (Van de Ven, Delbecq, and Koenig, 1976, p. 322). In software development, it means that different people and subunits working on a common project agree to a common definition of what they are building, share information, and mesh their activities. They must have a common view of what the software they are constructing should do, how it should be organized, and how it should fit with other software systems already in place or undergoing parallel development. To build the software efficiently, they must share detailed design specifications and information about the progress of software modules. In sum, they must coordinate their work so that it gets done and fits together, that it isn't done redundantly, and that components of the work are handed off expeditiously.

Characteristics of Software Development

Achieving a successful software system requires tight coordination among the various efforts involved in the software development cycle. Yet this coordination is difficult to achieve. As Curtis, Krasner and Iscoe (1988) note in their study of large software development projects, communication bottlenecks and breakdowns are very common. Indeed, several characteristics of software development make these coordination problems inevitable, not just common (Brooks, 1975).

Scale. A fundamental characteristic of many software systems is that they are very large and far beyond the ability of any individual or small group to create or even to understand in detail. If a software system were small, effective coordination could occur because a single individual or small group could direct its work and keep all the

implementation details in focus. But this ideal is impossible for many large software systems, where system size is measured in millions or tens of millions of lines of code.[1] Assuming typical productivity (measured in lines of new or changed production- quality code per staff year), a software system with one million lines of code might require 500 staff-years of effort (Martin, personal communication), summing across the analysts, programmers, support staff, testers, document writers, managers, and administrative infrastructure involved in a large project.

Efforts of this scale invariably lead to specialization and a division of labor. These organizational responses in turn lead to compartmentalization of interdependent actors through geographic, organizational, and social boundaries. Within these boundaries, unique subgroup perspectives, cohesiveness, ethnocentrism, and unwillingness to trade information increase (e.g., Brewer and Kramer, 1985; Tajfel, 1982). Barriers -- geographic, organizational, or social -- reduce people's opportunity and eagerness to share information and to learn from distant colleagues (Newcomb, 1961; Faunce, 1958; Monge and Kirste, 1980, Dearborn and Simon, 1958; Jablin, 1979; Barnlund and Harland, 1963; Sykes, Larntz, and Fox, 1976; Monge, Rothman, Eisenberg, Miller, and Kirste, 1985). While compartmentalization promotes organizational efficiency in large groups by shielding people from unnecessary information, it nonetheless creates new coordination tasks. Changes in one "compartment" require adjustments in others. Compartments limit people's breadth of experience, leading to errors, narrowness, and insufficient opportunity for comparing knowledge. They can reduce the motivation to interact with relevant others and to accept new ideas.

Uncertainty. The inherent uncertainty in software development compounds the coordination problems produced by large scale alone. By uncertainty, we mean the unpredictability of both the software and the tasks that software engineers perform. Unlike manufacturing, software development is a nonroutine activity, and the specification of what a software system should do changes over time (Brooks, 1987; Fox, 1982; Curtis, Krasner, and Iscoe, 1988). In part, change in software specification comes about because the external world that the software was designed to support also changes, as business needs, user desires, computer platforms, input data, and the physical world itself all change. The likelihood of change is greater whenever software is used directly by people, because it is often only by using software that purchasers and users understand its capabilities and limitations. Especially as they use software in circumstances for which it wasn't designed explicitly, the users demand new capabilities that had not been envisioned during the design.

Software development also is uncertain because specifications for it are invariably incomplete. Incompleteness partially results from limited domain knowledge and division of labor typical of software projects (Curtis et al., 1988). Too few people

1. The software to run ground control for the Apollo spacecraft in the 1970s contained about 23 million lines of code (Fox, 1982). The code for AT&T's 5ESS switch is about 10 million lines of code. The software to allocate lines in a telephone network contains over 10 million lines of code. Each generation of software is typically larger than the one that preceded it.

working on a software project have sufficient knowledge about the domain in which they are working. A project group writing software for a heads-up display for pilots needs in-depth knowledge of aircraft and aviation, as well as knowledge of computer science. Typically, analysts with varying degrees of domain knowledge interview customers and users, and then write specifications for architects and designers. Relevant information is inevitably lost. Some of the users' needs will not be uncovered by the analysts, and some of the analysts' tacit knowledge will not be reflected in the specifications. Thus, a major coordination problem in software development is that at many points the information that software architects and programmers need to make decisions is not available to them through documents, although users, analysts, and others in the project may have the knowledge necessary for these decisions.

Finally, software is uncertain because the different subgroups involved in its development often have different beliefs about what it should do and how it should do it. For example, analysts translate users' needs into requirements for system capabilities. As a result, they often adopt the point of view of the software's users. On the other hand, designers and programmers often have a more internal focus, with an emphasis on ease of development and efficiency of operation. As more groups become involved in software development, disagreements among them inevitably increase. These differences in points of view must be resolved for coordination to succeed.

Interdependence. The large size and uncertainty in software work would be less of a problem if software didn't require a strong degree of integration of its components. Much software is built of thousands of modules that must mesh with each other perfectly for the software system as a whole to operate correctly. The recent disruption of the AT&T long distance network (Travis, 1990) shows how interactions among modules introduced while updating software to give it more capabilities can have disastrous, unanticipated consequences. The implication is that poor coordination between subgroups producing software modules could lead to failure in integrating the modules themselves.

Informal communication. Both practical experience and organizational theory suggest that previous efforts in software engineering cannot by themselves solve the coordination problems in large software projects. The combination of large size, uncertainty, and interdependence requires special coordination techniques that may not be necessary in more routine operations. At the risk of oversimplification, one can say that most proffered solutions to the software crisis have taken one of three approaches: (1) technical tools, ranging from new workstations to syntax-directed editors, to improve the productivity of individual developers, (2) modularization, both technical, such as object-oriented programming, and managerial, such as the organizational separation of the requirements, coding, and testing functions, to encapsulate the behavior of program elements and individual software professionals, thereby reducing needs for coordination, and (3) formal procedures, both technical, such as version control software and specification languages, and managerial, such as test plans, delivery schedules, and requirement documents, to control communication among development personnel.

These techniques are likely to be an incomplete solution. Tools to increase the

productivity of individual programmers by definition do not solve coordination problems. No matter how successfully layered architectures and structured programming methodologies reduce the number of interfaces between modules, different people with different perspectives still must agree on what is to be built and must fit together pieces of software. Consensus formation, information sharing and coordination dilemmas that don't show at one level surface at another. Finally, trends toward formalization, while necessary for some purposes, for other purposes may be a misguided attempt to apply routine procedures when they are not applicable.

Formal, informal communication systems best suit different types of activities. Formal and impersonal communication is useful for coordinating routine transactions within groups and organizations. Formal coordination mechanisms often fail in the face of uncertainty, which typifies much software work. Under these circumstances, informal communication may be needed for coordination (Ochi, 1980; Van de Ven, Delbecq, and Koenig, 1976; Daft and Lengel, 1986).

Prior descriptions of communication in organizations, although not looking at software development per se, have shown the heavy and effective use made of informal communication for exchanging information in research and development settings. By informal communication we mean personal, peer-oriented, and interactive communication. It is to be contrasted with more formal procedures, such as written specification documents, formal specification languages, and automated reporting and tracking of program errors, in which communication is through documents and is unidirectional. The major findings are easy to summarize. First, informal, interpersonal communication is the primary way that information flows into and through research and development organizations (e.g., Adams, 1976; Allen, 1977; Tushman, 1977). Second, in the world of research and development as in many other domains (Culnan, 1983; Zipf, 1949), the ease of acquiring information is at least as important as the quality of the information in determining the sources that people use. Therefore physical proximity of the source is a major constraint on engineers' work-related information (Allen, 1977). Third, getting information and coordinating activity through informal, interpersonal communication is valuable both for individuals and for their organizations, especially as R&D tasks become more uncertain (Pelz and Andrews, 1966; Tushman, 1977).

The previous discussion points to a major and perhaps unresolvable tension in large software development projects. Because of interdependence, different groups involved in a software development project must be tightly coordinated. Because of the high degree of uncertainty typical of software projects, informal, interpersonal communication is a valuable method for achieving this coordination. But because of the large size of these projects, the inefficiencies of pairwise face-to-face communication may preclude its use as a major technique for solving coordination problems in large project groups.

The present paper is an empirical attempt to examine the conditions under which different techniques are used for coordinating software development and the conditions under which various of them actually succeed or fail in improving coordination. For the reasons just described, we concentrate on the contrast between more formal, impersonal techniques, such as design and requirements documents and status tracking

methodologies, and more informal, interpersonal techniques, such as peer discussion and unstructured electronic mail. Structured, interpersonal meetings, such as status reviews or design reviews are intermediate on the formality dimension.

SURVEY STUDY OF COORDINATION IN SOFTWARE DEVELOPMENT

We surveyed the intergroup coordination practices in one large software development company. The survey focused on three factors: 1) coordination practices used, 2) structural characteristics of projects that might interact with the practicality and utility of various coordination techniques, and 3) the success of the projects on several dimensions. We were particularly interested in features of the projects and the coordination practices that influenced the sharing of information and of goals.

The research site. The research site was the software development divisions of a research and development company. They employed approximately 3,000 managers, analysts, software engineers, programmers, testers, and documentation specialists. Collectively, the staff worked on the development of a wide range of products for the telecommunications industry, using a wide range of techniques in the development process. In terms of scale, they ranged from two to four person projects with development on PCs to large main-frame systems, with 14 million lines of code already developed and 150 people on staff at once. The median project had over 15 people on staff at the time of the survey. In terms of software life cycle, the projects ranged from those in the specification stage with active negotiations with clients and with other development organizations to more maintenance-oriented projects, where new releases were meant to fix bugs and add small numbers of features. While all projects used both formal and informal communication to coordinate activity, the balance differed across projects. For example, on the more formal end, requirement specification documents were written by members of a different vice presidential area than those who design software architecture. Information about the needed software capabilities was conveyed through these formal specification documents. In other projects representing the more informal end of the continuum, the responsibility for assessing software capabilities and for actually writing the code resided within the same 30-person department and communication was through informal, interpersonal contacts supplemented by sketchy requirements documents. Similarly, some departments made extensive use of electronic mail and bulletin boards to distribute project knowledge, while others did not use these facilities.

Sample. The sample consisted of 150 supervisory groups involved in some aspect of software development, representing approximately 80 different software systems. The coordination survey was sent to 750 people (150 first level supervisors and 600 technical staff). We included all phases of software from requirements to field support. Eighty-eight percent of the sample returned the questionnaire after three mailings and of these 563, or 75% of the total sample, returned usable data.

There were 65 projects for which at least two people provided data. Projects with only one respondent were dropped. The number of respondents per project ranged from 2 to 47 with a mean of 7.6 and median of 4. Depending on the question, analyses are based on

563 individuals or 65 projects.

Measures. The survey measured the following aspects of software development within a project. Table 1 shows examples of all measures.

1. **Structural characteristics of projects.** These include *project size, project age,* and *stage* in the software life cycle. Two other structural characteristics were especially important. One, *organizational interdependence,* is the extent to which members of a project get inputs from and pass output to other groups within the company that are outside of their immediate supervisory group. The other, *project certainty,* is the extent to which a project is stable and the work consists of tasks which are well understood and easily accomplished by local expertise.

Insert Table 1

---

2. **Coordination techniques used.** These include: *formal, impersonal coordination techniques,* such as written requirements documents, modification request tracking, and data dictionaries; *formal, interpersonal techniques,* such as requirement reviews, status reviews, and code inspections; and *informal interpersonal techniques,* such as unscheduled group meetings or co-location of requirements and design staff. Among the informal techniques, we examine separately *electronic communication,* such as electronic mail and electronic bulletin boards, and *interpersonal networks.* The interpersonal network measure requires additional explanation. Respondents were given a list of 100 first-level supervisors randomly selected from the software development divisions and circled all those with whom they had talked the previous 2 years. Projects in which people had many contacts outside the project had extensive interpersonal networks.

3. **Outcomes of coordination techniques.** This paper uses two measures. The *informed scale* measures respondents' assessments of how adequately informed they and the project's managers were about project status and responsibilities. The *coordination success scale* measures respondents' assessments of how well their projects were going and integrating with the work of other organizations. (See Table 1.)

RESULTS

Techniques for spreading project information and coordinating work. We examine the conditions under which projects used formal procedures and informal ones to coordinate work and the value they judged these procedures to have. For each technique respondents made two ratings: (1) the extent to which the project used the technique to get work done and (2) the value of each technique for spreading information and coordinating work regardless of its use.[2] Each rating was made using a 7 point scale with

"1" indicating "no use" or "low value" and "7" indicating "used a lot" or "high value". Analyses are based on linear regression equations predicting techniques used from characteristics of projects, and their perceived value from project characteristics and use.

As Table 2 shows, projects tended to use formal impersonal procedures such as project documents, milestone and delivery schedules, and error tracking more when the projects were certain, larger, and had passed the requirements and design stages of their life cycles. These formal procedures were judged valuable the more they were used and were judged especially valuable when projects were more certain. Formal information exchange meetings such as requirements and status reviews were used most in large projects. They were judged most valuable when projects were more certain and when they were in planning stages. On the other hand projects tended to use informal, interpersonal communication such as unscheduled meetings very frequently, and used it regardless of project size, certainty, or life cycle. This informal communication was judged especially valuable when the project was certain and when it was in the planning stages. Finally, electronic communication was used as frequently as the formal, impersonal procedures, and was used more when projects were heavily dependent on input from other groups in the company. No project characteristics predicted the value of electronic communication, once one controlled for its use. Finally, projects had more extensive interpersonal networks outside the project when the project was smaller, when it depended was interdependent, and when it was certain.

Insert Table 2

Figure 1 extends these results by looking at the use and value of particular techniques. Since people tend to judge techniques they use as more valuable, our analysis attempts to correct for this inherent bias. Figure 1 shows the line found by regressing the use of various techniques on value. Those techniques that are above the regression line were judged to be more valuable than predicted by current use, while those below the line were judged to be less valuable than one would expect based on their use. Techniques that were judged to be statistically significantly more valuable than one would predict based on their use are marked with a "+". These include both informal discussions with geographically local colleagues (discussions with boss, discussions with peers, group meetings) and other interpersonal procedures that make available points of view external to ones local work group. Thus requirements reviews, design reviews, and customer testing allow personnel with different responsibilities and points of view, such as requirements analysts, systems engineers, architects, programmers, testers, and customers, to comment on each others work interactively, but in structured a setting. The co-organization of requirements and development, -- the placement of these functions

2. Note that the procedure was different for the interpersonal networks measure, described above. For this measure, respondents completed a communications roster, rather than estimating how frequently they used interpersonal networks. They did not rate the value of the networks.

under a single line of management -- allows for more frequent communication between the personnel responsible for these functions. Techniques that respondents judged to be statistically significantly less valuable than the amount they were used are marked with a "-". These tended to be formal coordination techniques, both interpersonal ones in which the information conveyed is relatively routine (code inspections, status reviews) and impersonal ones (project documents, source code, milestones, and error tracking).

Insert Figure 1

How do software professionals get help? The way project members deal with specific work problems also reflects coordination patterns in software development. We asked people to describe the most recent project-related problem they had that they couldn't solve alone. A variety of technical and managerial problems were reported. Technical problems included such difficulties as dealing with corrupt data in a database, deciding on a new programming language to use on a project, determining whether a particular piece of data was needed for an interface, or investigating why a software module ran too slowly. More managerial problems included specifying a human interface that was being jointly defined by two separate companies, calculating a complete cost estimate for a new work project, or handling a conflict in responsibilities between a developer and a documenter.

From a list of information and consultation sources, respondents rated (1) the extent to which the source was used to solve the particular problem and (2) the potential usefulness of the source, whether or not it was actually used for the particular problem. Again, ratings were made on a 7-point scale with "1" indicating that the source was "not consulted" or was of "low" potential usefulness, and "7" that the source was "strongly consulted" and of "high" potential usefulness.

Figure 2 parallels Figure 1 and shows which information sources are perceived to be underused (above the line), and which are perceived to be overused (below the line).

By far the predominant source of information was other people. Consistent with previous research (e.g., Allen, 1977), software engineers overwhelmingly get their information from other people and the ease of getting the information is a critical determinant. Thus, in our data, three of the top four information sources were other people. Other project members, often in close physical proximity to the respondent, were used as information sources far more than other source. On the other hand, respondents judged that people from outside the project and therefore difficult to access (other company employees not on the project, and subject matter experts from outside the company) were perceived as significantly underused relative to their value. All forms of written documentation were perceived as less valuable than personal contacts, and some of them were judged to be significantly overused relative to their value (books, and journals, electronic bulletin boards, and project documentation). Again, these results suggest the value of getting information by interpersonal means from outside ones immediate work group.

Insert Figure 2

Predicting Project Coordination Success. What project characteristics and coordination techniques predict successful at intergroup coordination? To answer this question, we used path analysis. Path analysis uses a system of simultaneous linear regression equations to test causal hypotheses by holding constant the effects of antecedent variables when estimating the causal impact of a focal variable.[3] Figure 3 presents the significant relationships among some of the variables from Table 2 and the outcome measures; the numbers on the arcs are the standardized beta weights.

The first conclusion to be drawn from Figure 3 is that projects with different characteristics differ in the degree to which they were coordinated. As one might be expect, projects that were older, smaller, and less interorganizationally interdependent were better coordinated. In addition, projects that were more technically certain (i.e., stable, homogeneous and confronting routine problems) had project staff who were better informed and were better coordinated. Interestingly, both project age and project interdependence had part of their effects on coordination success through project certainty. That is, older projects are on average more stable and homogeneous, and these factors make coordination easier to accomplish. Similarly, projects that are less interdependent have more control over the directions they set and the resources they have available. These factors make project members more informed about decisions that affect them and project success. These factors also make projects more certain and in turn better coordinated.

We had seen earlier that projects with different characteristics rely on different coordination techniques. In particular, we have seen that larger projects, more certain projects, and projects that have passed beyond the planning phases of the software life cycle were more likely to use formal, impersonal coordination techniques such as written documents, milestone and delivery schedules, and management tracking of errors and change requests. The interesting finding revealed in Figure 3 is that the use of formal procedures does not seem to improve intergroup coordination once one controls for the conditions under which they are used.

Figure 3 shows that different factors predict whether members of a project have a dense interpersonal network that extends beyond the project's boundaries. (See also Table 2.) In projects that are highly interdependent, members of the project by necessity know

---

3. In the path analysis, we assumed the following casual ordering: Structural characteristics, such as project age, size, and interdependence could potentially influence a project's certainty. Together these project characteristics could potentially influence the coordination techniques a project adopts, which in turn, could potentially influence how well informed project members were and how successfully the project was coordinated. This causal ordering is consistent with both the prior literature on task certainty and coordination (e.g., Perrow, 1970; Tushman, 1977) and the literature in software engineering (e.g. Brooks, 1975; Fox, 1982). However, like all path analyses based on cross-sectional data, the ordering can be debated. For example, the lack of success on a project could cause managers to increase its size (one of the dangers Brooks warned about), or the coordination techniques that a project adopts may cause the project to become more or less certain.

many others outside of their projects. On the other hand, in large projects the interpersonal network is within the project and as a result project members know few people outside of their project. Finally, more certain projects have more extensive interpersonal networks.

The maintenance of an extensive interpersonal network improves outcomes. In particular, in projects in which members talk to others outside of the project, both project members and their management know more about project status and commitments. They are rarely caught by surprise. This greater awareness aids their intergroup coordination.

Perhaps the most interesting finding revealed in Figure 3 is the the statistical interaction between project certainty and interpersonal networks. This interaction means that the beneficial effect of interpersonal networks was most pronounced when projects were uncertain. This finding is consistent with organizational research, reviewed in the introduction, that informal, interpersonal communication is necessary for coordination under conditions of uncertainty.

Insert Figure 3

SUMMARY

The results show the value of interpersonal communication, both informal and more formal, as mechanisms to achieve coordination in software development. Software development requires personal communication across functional boundaries to cope with uncertainty. Managerial and technical problems continuously arise in the process of creating software. While people can solve some of these problems by themselves or by examining static documents, others demand information or cooperation from other people. An employee who doesn't understand the work flow that software was designed to support, who needs to negotiate which of two modules to change to fix a bug, or who needs to fill in the gaps left in a functional specification document, needs help from someone else in the software development process.

The standard response when one confronts a problem that cannot be solved alone is to go to a colleague close by. However, not all the necessary knowledge or relevant viewpoints can be gathered from local colleagues, who are often consulted more from convenience than from relevance or competence. Personal contact with those outside of ones immediate project is one mechanism to get this information. Thus, projects with denser cross-boundary networks were better informed and coordinated, and extensive personal networks were particularly useful in uncertain projects.

However, personal contact between organizations through interpersonal networks is not a panacea for coordination problems in large software projects, both because of the excessive transactions costs resulting from the many pairwise conversations and because of the ephemeral nature of the information transferred in them. Thus, a large project will also need institutionalized communication events, like requirements reviews, design

reviews, and customer testing, where the diverse groups involved in the project can come together in a controlled way and from which archival records can be derived. In the present study, project members judged these between-organizational meetings were underused and supported relative to their value. On the other hand, project members did not call for increased use of meetings that typically exchange routine information or that don't involve outsiders, such as status review meetings or code inspections.

In the company described here, impersonal modes of coordination, such as project documents, memos, and bulletins, milestone and delivery schedules, and error tracking, were a standard part of the software development process and hence were heavily used in most projects. Surprisingly, once one controlled for the conditions under which they are used, namely in larger, more certain projects that have passed the planning stages, greater reliance on them did not lead to better coordination. Neither did project members call for more use of these procedures. These findings suggest some of the limitations on formal procedures.

The research reported here is consistent with a large literature about coordination in organizations. In some ways, coordination in software development differs little from coordination in hospitals (Georgopoulous & Mann, 1962), social service agencies (Van de Ven, et al, 1976), or research and development more generally (Tushman, 1977). Yet concluding that the problems of software engineering are common managerial ones requires stronger evidence. In particular, the present study suffers from two weaknesses. The first is the simplified assumptions about causal ordering of variables, referred to previously. For example, while most organizational theorists treat task certainty as a fundamental property of organizations that influences the coordination techniques used and valued (Perrow, 1970; Katz & Tushman, 1978), the causation is undoubtedly bidirectional, and organizational responses also change certainty. The second weakness is the nature of the outcome measures reported here. They are self-reports from technical workers and first level supervisors. One would like similar results coming from a wider variety of metrics, including the quality of the software, time to delivery, and customer satisfaction, before confidently endorsing them.

Improving the process. How might software projects gain some of the advantages of interpersonal communication for coordination under uncertainty, while minimizing the overhead and lack of managerial control that pairwise communication entails? Better communication support could be achieved either through organizational means or through technological means. We review both approaches below.

Respondents in the present study thought that coordination could be substantially improved if different functional specialties, such as requirements analysis and code development, were located in the same organization or were physically close to each other. These changes would make others with different information or priorities more accessible and also provide a managerial mechanism to resolve differences in goals and point of view. Research in domains analogous to software development suggests that these approaches do indeed work to improve the design and development process. For example, Clark, Chew, and Fujimoto (1987) determined that differences in managerial structures and communication patterns accounted for a sizable proportion of the

advantage that Japanese car manufacturers have over European and American ones in terms of the time to bring a new model to market and the engineering hours used to develop it. To oversimply their complex study, Japanese automobile design and engineering typically had smaller, multifunctional project teams, strong project managers with the expertise and power to control and make technical decisions, and intense formal and informal lateral communication between functional specialties. On the other hand, US manufacturers tended to have larger projects, weaker project managers to coordinate but not make technical decisions, and more formal between-specialty communication through frequent large meetings. The Europeans typically used a functional organization in which coordination was achieved through formalized rules and procedures, no project manager and intense informal communication across functional boundaries.

While the analogy between automobile and software design and manufacturing is not perfect[4], the Japanese managerial and communication models bear a close resemblance to the chief programmer teams advocated by Weinberg (1971) and chief architect model that Curtis et al (1988) identified as characteristic of successful software projects. The Japanese example suggests that both of these software engineering visions of strong, technical managerial control must be supplemented by intense lateral communication between functional specialities.

These findings seriously undermine the traditional waterfall model of software development in which each hierarchically organized, functionally specialized group (be it requirements or system test) hands off its products to the next downstream group. Functional organization coupled with over-reliance on formal coordination procedures leads to larger development teams and longer software life cycles. An error made "upstream" cannot be caught until the next stage in the process. Functionally organized software efforts tend to populate each stage with specialists while strong team leader projects tend to staff more broadly.

As the Japanese case of automobile production points out, successful projects rely on both formal and informal communication techniques. Respondents in this survey found some meetings, such as design and requirements reviews, especially valuable for coordination. These meetings must make and review highly consequential but uncertain decisions, and must be informed by a wide variety of personnel. Decisions reached at such meetings have important consequences for the project as a whole, and decisions made at them as well as the reasoning behind them should be a permanent, retrievable part of the project memory. This suggests that ways to automatically capture and document these decisions are important. Technology could be summoned to provide better support for these meetings. For example, some of these meetings could be conducted through computer conferences or using computer-based group decision support systems (see Kraemer and Pinsonneault, 1990, for a review). Computer-

4. For example, software development is more uncertain with changing platforms, languages and applications, while automobile manufacture requires more cross organizational interdependence, with substantial outsourcing of components.

supported communication could have the consequence of opening up the discussion to more and more divergent points of view, improving the quality of decisions reached, and providing a content addressable archive of the meeting.

Information technologies could be made more valuable for supporting informal communication in software development, if two major social problems could be solved. The first is the difficulty of insuring that the relevant information that some project members know is made explicit and available for others to use. One of the values of personal networks is that people will contribute information when explicitly asked, often going well beyond the original question. However, in general, people under-contribute information when the information is intended for the anonymous, common good (e.g., explanatory comments in source code). This tendency is a natural consequence of the economics of common goods, in which ones personal interest is best served by receiving from but not giving to the common good (cf Hardin's, 1968, discussion of the common's problem and Thorn and Connolly's, 1990, application to discretionary databases). The second difficulty is one of coping with information overload. Given the vast amount of information associated with software projects, it is likely that individuals will be bombarded with information, both relevant and irrelevant. It is not clear whether standard information retrieval techniques can deal with this flood.

REFERENCES

1. Adams, J. S. The structure and dynamics of behavior in organizational boundary roles. In M. D. Dunnette (Ed.), *Handbook of Industrial and Organizational Psychology*. Chicago: Rand-McNally, 1976, 1175-1199.
2. Allen, T. J. *Managing the Flow of Technology*. Cambridge: MIT Press, 1977.
3. Bamlund, D. C. and Harland, C. Propinquity and prestige as determinants of communication networks. *Sociometry 26*, 1963, 466-479.
4. Brewer, M. B. and Kramer, R. M. The psychology of intergroup attitudes and behavior. *Annual Review of Psychology 36*, 1985, 219-243.
5. Blau, P. and Scott, W. R. *Formal organizations*. San Francisco: Scott, Foresman, 1962.
6. Blazer, R. Imprecise program specification. Report ISI/RR-75-36, Information Science Institute, December, 1975.
7. Brooks, F. P. *The Mythical Man-Month*. Addison-Wesley, Reading, MA, 1975.
8. Brooks, F. P. No silver bullet: Essence and accidents of software engineering. *IEEE Computer Society 20*, (April, 1987), 10-18.
9. Burnham, D. The I.R.S.'s bumbling efforts to update its computers. *The New York Times*, April 8, 1990, F 12.
10. Clark, K. B., Chew, W. B., and Fujimoto, T. Product development in the world auto industry. *Brookings Papers on Economic Activity 3*, 1987, 729-781.
11. Curtis, B., Krasner, H., and Iscoe, N. A field study of the software design process for large systems. *Communications of the ACM 31(11)*, 1988, 1268-1287.
12. Culnan, M. J. Environmental scanning: The effects of task complexity and source accessibility on information gathering behavior. *Decision Science 14*, 1983, 194-206.
13. Daft, R. L., and Lengel, R. H. Information richness: A new approach to managerial behavior and organization design. In B. Staw and L. L. Cummings (Eds.), *Research in Organizational Behavior* (Vol. 6). Greenwich, CT: JAI Press, 1984.
14. Dearborn, D. C. and Simon, H. A. Selection perception: A note on the departmental identification of executives. *Sociometry 21*, 1958, 140-144.
15. Faunce, W. A. Automation in the auto industry: Some consequences for in-plant social structure. *American Sociological Review 23*, 1958, 401-407.
16. Fox, J. M. *Software and its Development*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
17. Georgopoulos, B.S. & Mann, F.C. *The Community General Hospital*. NY: Macmillan, 1962.

18. Hardin, G. The tragedy of the commons. *Science, 162*, 1962, 1243-1248.

19. Hunting the 'blueprint' of eternity. *Washington Post, April 8, 1990*, A1,26,27.

20. Jablin, F. M. Superior-subordinate communication: The state of the art. *Psychological Bulletin 86*, 1979, 1201-1222.

21. Katz, R. & Tushman, M. Communication patterns, project performance, and task characteristics: An empirical evaluation in an R&D setting. *Organizational Behavior and Human Performance, 23*, 1978, 139-162.

22. Kraemer, K. & Pinsonneault, A. Technology and groups: Assessment of the empirical research. In J. Galegher, R. Kraut & C. Egido, (Eds.). *Intellectual teamwork: Social and technological foundations of* cooperative work. Hillsdale, NJ: Lawrence Erlbaum, Associates, 1990.

23. Martin (personal communication, 1990).

24. Monge, P. R., Rothman, L. W., Eisenberg, E. M. , Miller, K. L. and Kirste, K. K. The dynamics of organizational proximity. *Management Science 31*, 1985, 1129-1141.

25. Newcomb, T. R. *The Acquaintance Process.* New York: Holt, Rinehart and Winston, 1961.

26. Ouchi, W. G. Markets, bureaucracies, and clans. *Administrative Science Quarterly 25*, 1980, 129-140.

27. Pelz, D. C. and Andrews, F. M. *Scientists in Organizations: Productive Climates for Research and Development.* New York: Wiley, 1966.

28. Perrow, C. *Organizational Analysis: A Sociological View.* Belmont, CA: Wadsworth, 1970.

29. Sykes, R. E., Larntz, K., and Fox, J. C. Proximity and similarity effects on frequency of interaction in a class of naval recruits. *Sociometry 39*, 1976, 263-269.

30. Tajfel, H. Social psychology of intergroup relations. *Annual Review of Psychology 33*, 1982, 1-39.

31. Thorn, B. K. & Connolly, T. Discretionary data bases: A theory and some empirical findings. *Communications Research, 14*, 1987, 512-528.

32. Travis, P. Why the AT&T network crashed. *Telephony 218(4)*, (January 22, 1990), 11.

33. Tushman, M. L. Special boundary roles in the innovation process. *Administrative Science Quarterly 22(4)*, 1977, 587-605.

34. U.S. General Accounting Office. *Contracting for Computer Software Development--Serious Problems Require Management Attention to Avoid Wasting Additional Millions.* U.S. Department of Commerce, National Technical Information Service, P880-105638, Washington, D.C., 1979.

35. Van de Ven, A. H., Delbecq, A. L., and Koenig, R. Jr. Determinants of coordination modes within organizations. *American Sociological Review 41*, 1976, 322-338.

36. Weinberg, G. M. *The Psychology of Computer Programming*. New York: Van Nostrand Reinhold Co, 1971.

37. Zipf, G. K. *Human Behavior and the Principle of Least Effort*. Cambridge, MA: Addison-Wesley, 1949.

STRUCTURAL CHARACTERISTICS

PROJECT SIZE

1. Number of people working on this project across the company.

PROJECT AGE

1. Maximum number of years that any project member worked on the project.

PLANNING STAGE

1. Percent of project staff having as their major work activity either requirements analysis and specification or high level software architecture and design.

ORGANIZATIONAL INTERDEPENDENCE (5-items, Alpha = .69)

1. Extent to which your work is interrelated with people in your division, but outside your group
2. Extent to which work is related with the work of others outside your division, but within the assistant vice presidential area
3. Extent to which work is related with work outside the assistant vice presidential area, but within the vice presidential area.

PROJECT CERTAINTY (8-items, Alpha = .65, Adapted from Van de Ven, Delbecq, and Koenig, 1976)

1. Clearly defined body of knowledge or subject matter guiding work on the project
2. Extent to which people in a particular district do the same type of work
3. Stability of the detailed specifications for the project

COORDINATION TECHNIQUES

FORMAL, IMPERSONAL PROCEDURES (5-items, Alpha = .63)

1. Project documents and memos
2. Project milestones and delivery schedules
3. Modification request and error tracking
4. Data dictionaries

FORMAL, INTERPERSONAL PROCEDURES (5-items, Alpha = .74)

1. Status review meetings
2. Design review meetings
3. Code inspections

INFORMAL, INTERPERSONAL PROCEDURES (2-items, Alpha = .56)

1. Group meetings
2. Co-location of requirements and development staff

ELECTRONIC COMMUNICATION (2-items, Alpha = .38)

1. Electronic mail
2. Electronic bulletin boards

INTERPERSONAL NETWORK

1. Number of supervisors from outside the project talked to in the previous two years.

OUTCOMES

PROJECT MEMBERS INFORMED (5-items, Alpha = .83)

1. Management on the project has an accurate view of how well the project is going
2. Immediate manager is a good source of information about relevant work from other areas of company
3. People are well informed about project status

COORDINATION SUCCESS (5-items, Alpha = .65, Adopted from Georgopoulous and Mann, 1962)

1. Lack of duplication and redundancy in the work of groups on the project
2. Extent to which the group avoids working in a "crisis mode"

TABLE 1. Survey measures, with examples

	Coordination Techniques								
	Formal impersonal procedures		Formal interpersonal procedures		Informal interpersonal procedures		Electronic communication		Interpersonal networks
	Use	Value	Use	Value	Use	Value	Use	Value	
Project characteristics									
1. Project age	0.17	0.04	0.13	0.14	0.16	-0.03	-0.03	-0.06	-0.08
2. Project size	0.32*	0.02	0.50*	0.08	-0.20	0.07	0.21	0.10	-0.30*
3. Planning stage	-0.30*	-0.01	-0.05	0.36*	0.27*	0.28*	0.14	0.18	0.14
4. Group interdependence	-0.02	0.10	-0.18	-0.18	-0.01	-0.02	0.34*	-0.03	0.42*
5. Project certainty	0.24*	0.39*	0.16	0.32*	-0.08	0.46*	0.10	0.06	0.29*
Use of the coordination technique	-	0.56*	-	0.40*	-	0.48*	-	0.77*	-

Note. Entries are the standardized beta weights in a regression analysis predicting the use and value of coordination techniques (columns) by project characteristics (rows).

Use of a technique was entered into the equation predicting its value. Interpersonal networks were measured differently from the other coordination techniques. A distinction between use and value was not made.

* $p < 0.05$
N = 65 projects

TABLE 2. Predicting the use and value of coordination techniques.

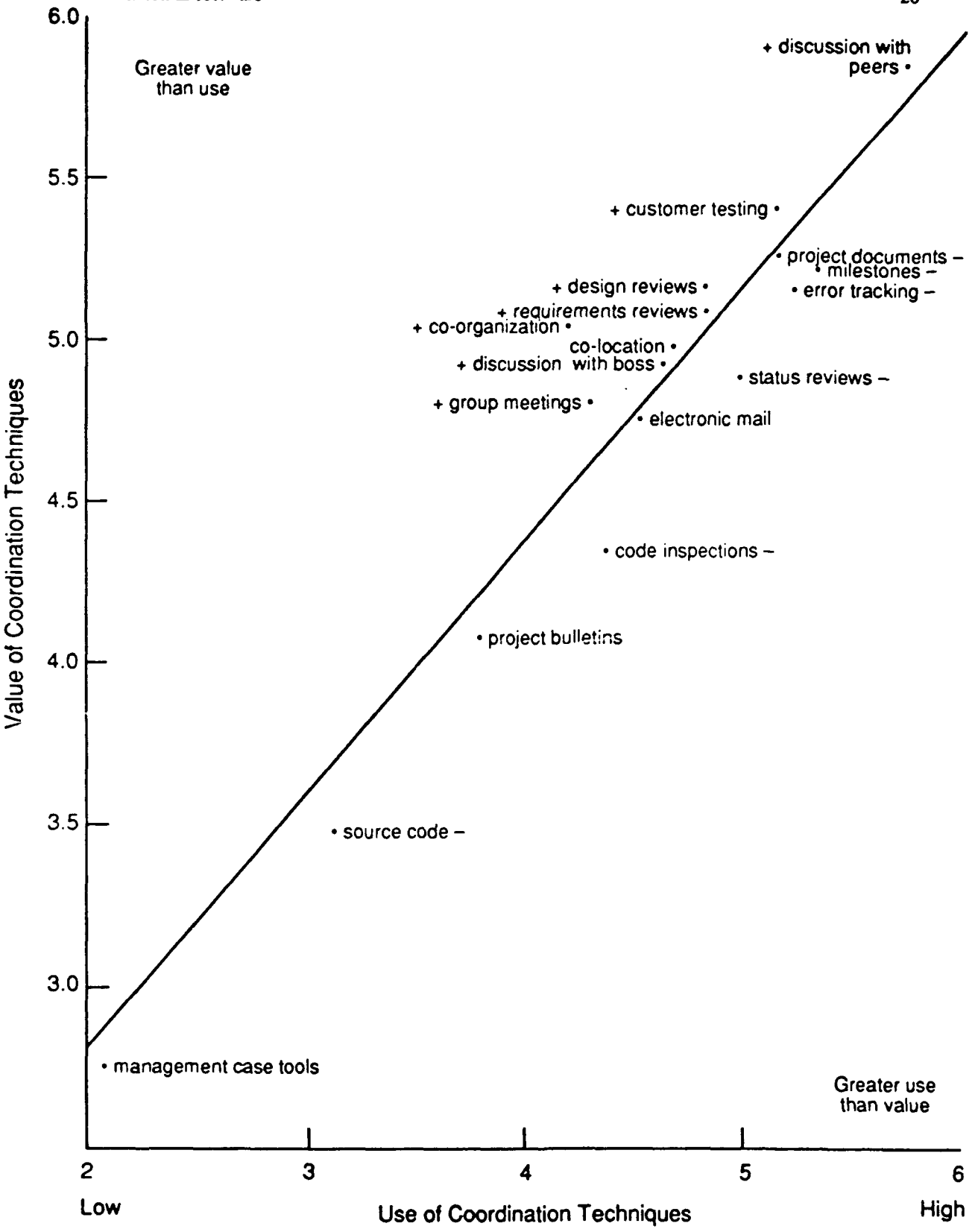


Figure 1. Comparing the use and value of coordination techniques

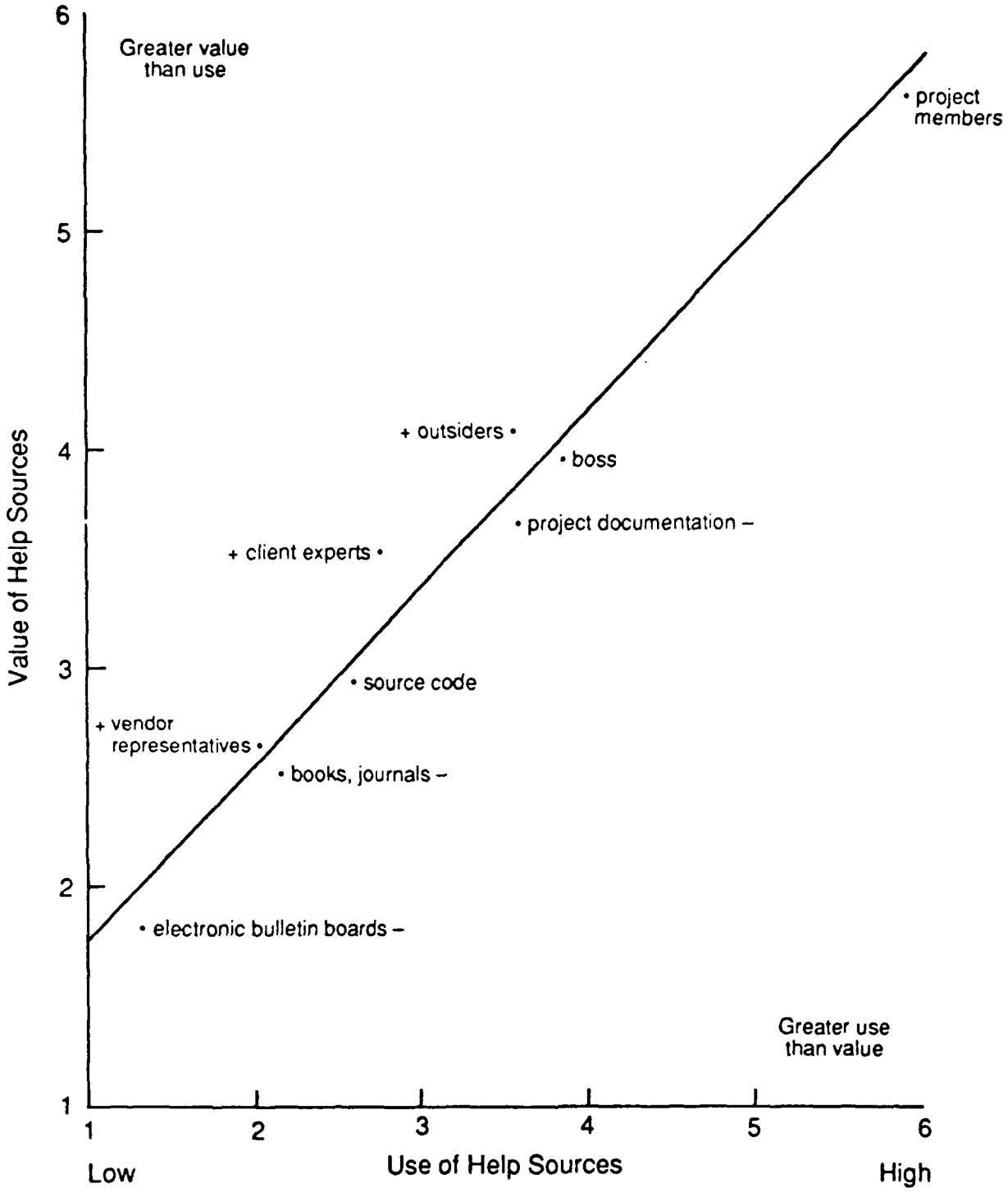
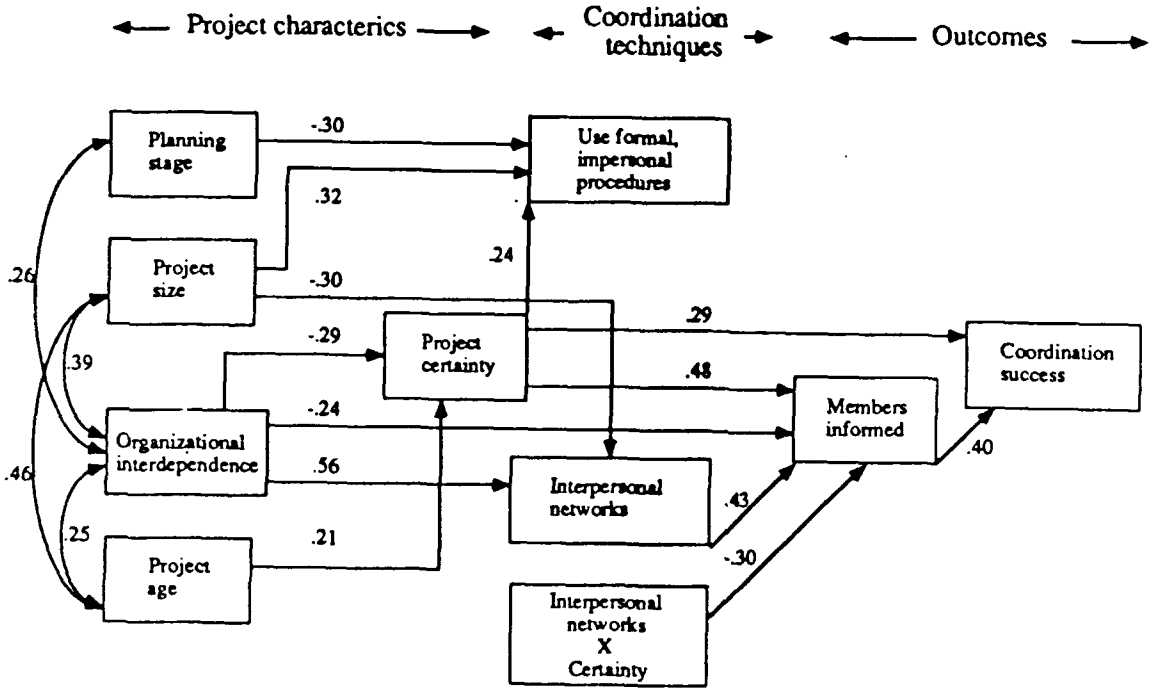


Figure 2. Comparing the use and value of sources of help



Factors influencing coordination

Figure 3. Factors influencing successful coordination